

# Manual de Integração e Simulação OMNeT++ com OPC UA

Este manual assume a utilização de um sistema Linux com distro baseado em Debian (Ubuntu, Pop!\_OS, Kali Linux etc).

Todos os códigos fornecidos e citados estão no [repositório](#) do GitHub.

Após a instalação e configuração do software com o SDK OPC UA, é **extremamente** necessária a leitura da [documentação](#) oficial do OMNeT++ junto com o estudo dos tutoriais TicToc fornecidos.

## Índice

[Instalações e Configurações Iniciais](#)

[Documentação e Construção da Primeira Simulação](#)

[Documentação da Simulação entre 5 clientes e 1 servidor](#)

[Documentação da Simulação entre dois hosts e um switch utilizando o framework INET sem conexão OPC](#)

[Integração com GDS \(Global Discovery Server\)](#)

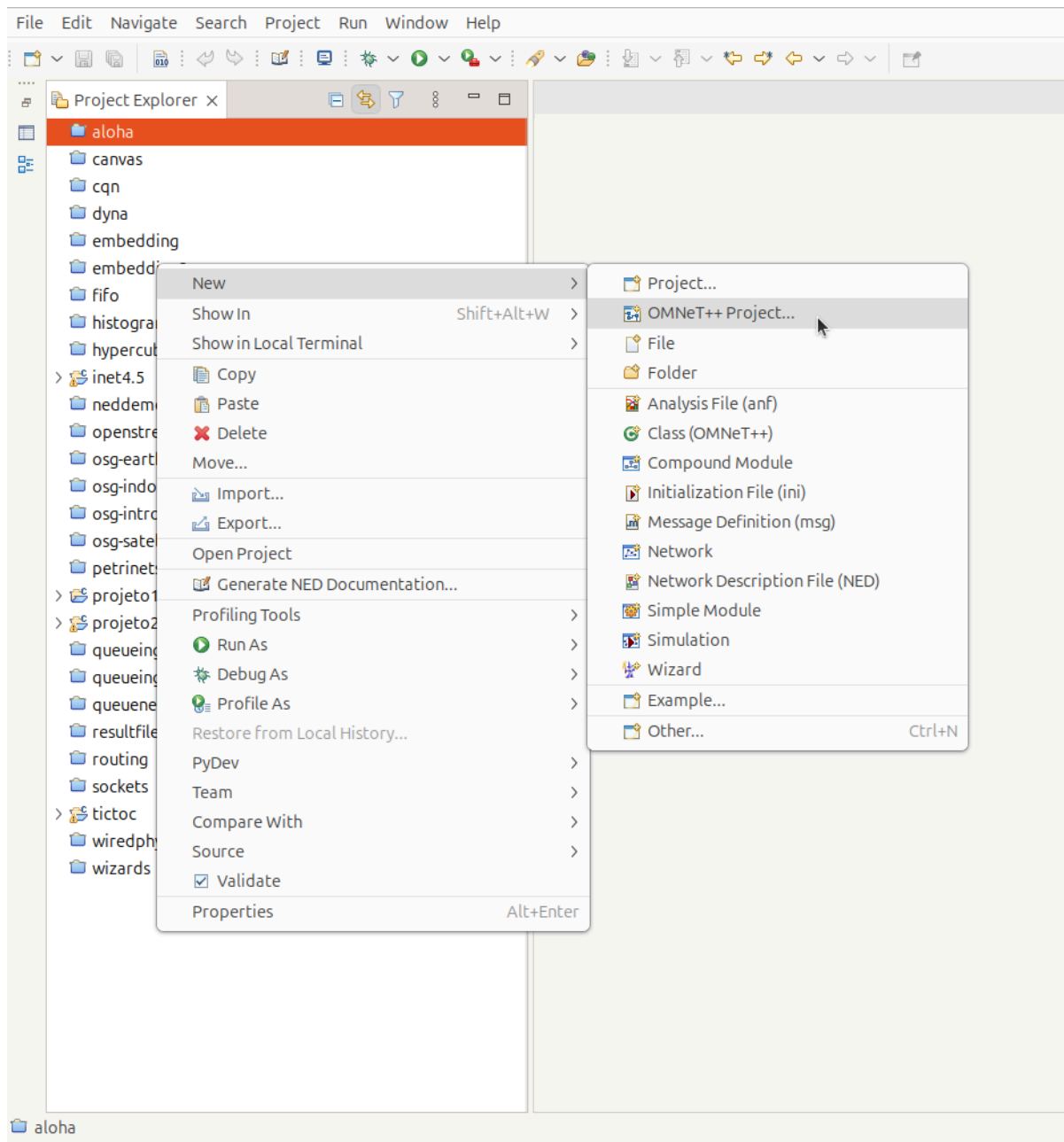
[Dificuldades e Próximos Passos](#)

## Instalações e Configurações Iniciais

Nesta seção será abordado como integrar o SDK OPC UA da Unified Automation com o OMNeT++.

- 1) Instale o OMNeT++ com base na documentação oficial: <https://omnetpp.org/>.  
Assumimos que a pasta de instalação esteja em '~/omnetpp-6.0.3'.
- 2) Faça o download do SDK OPC UA C++ da Unified Automation:  
<https://www.unified-automation.com/products/server-sdk/c-ua-server-sdk.html>
- 3) Extraia os arquivos do SDK para uma pasta, no nosso exemplo, está localizada dentro do diretório do OMNeT++: '~/omnetpp-6.0.3/opc'
- 4) Instale o CMake, OpenSSL, LibXML2:
  - a) `sudo apt install cmake`
  - b) `sudo apt install openssl`
  - c) `sudo apt install xml2`

- 5) Inicie o OMNeT++. Na aba Project Explorer, clique com o botão direito do mouse na área vazia, New -> OMNeT++ Project



- 6) Escolha um nome para o projeto, no nosso exemplo, “projeto”. Em seguida, clique em “Next”.

**New OMNeT++ Project**

Create a new OMNeT++ Project

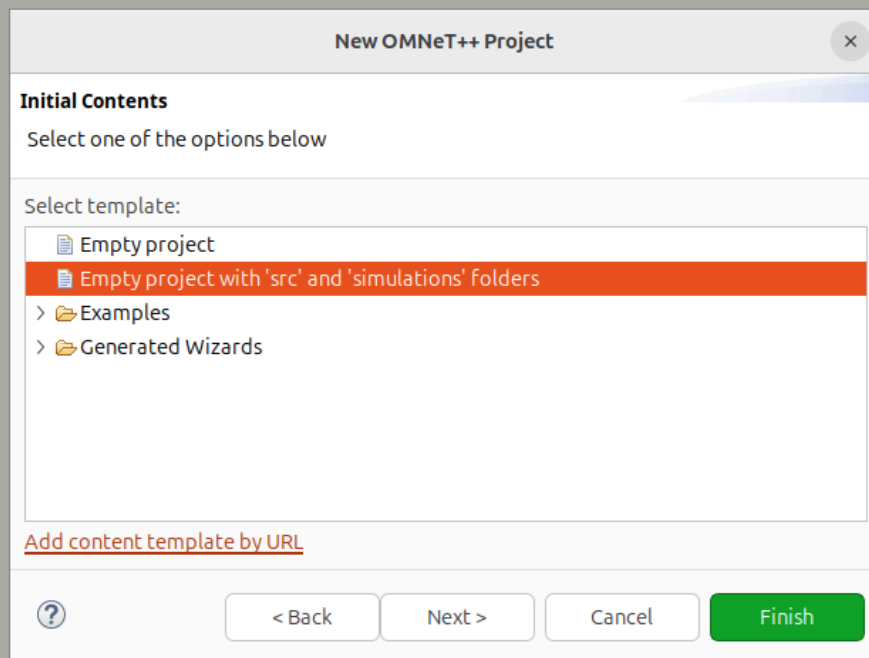
Project name:

☒ Use default location

Location:

☒ Support C++ Development

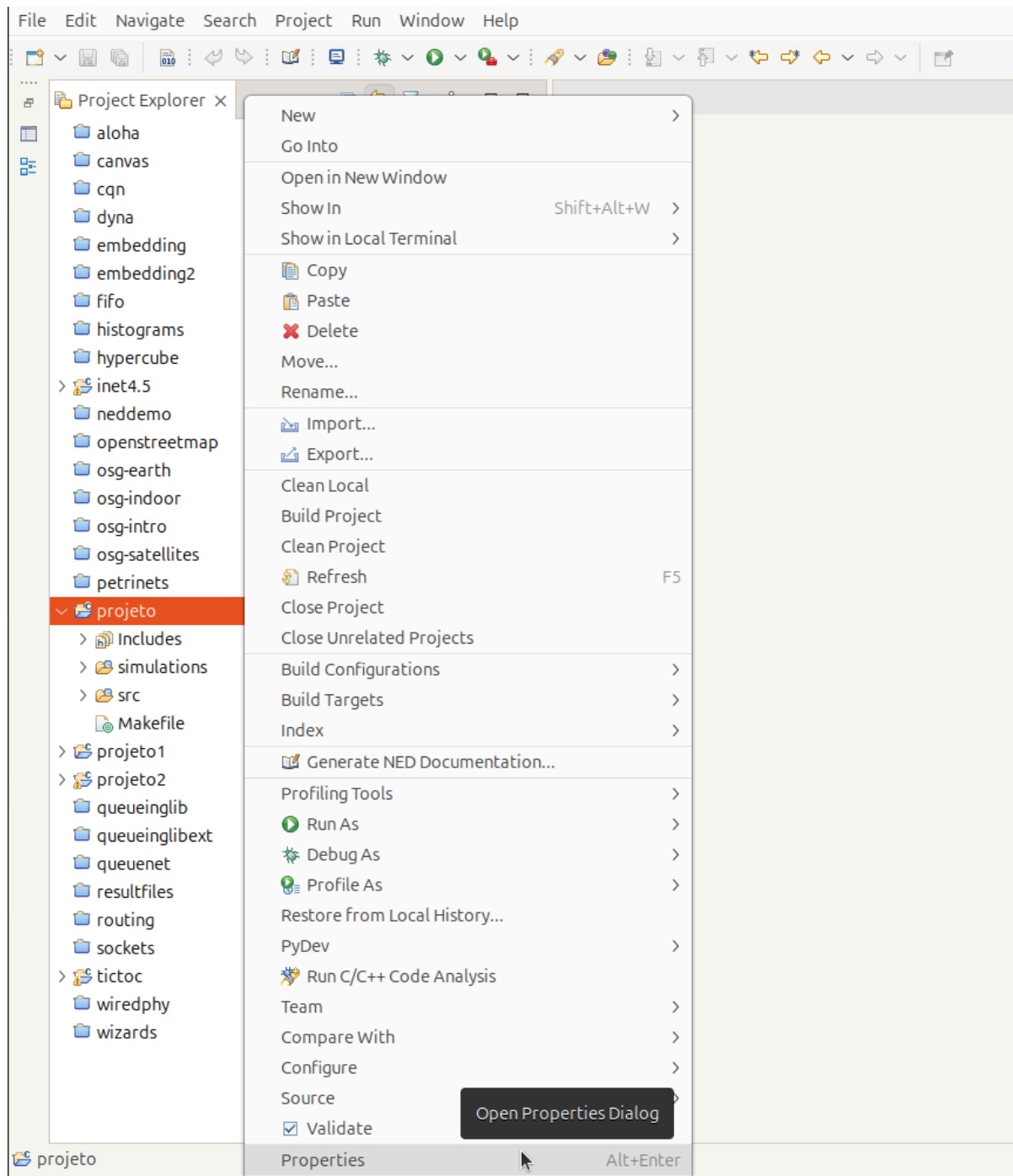
- 7) Escolha a segunda opção para que sejam criadas as pastas necessárias para a simulação. Em seguida, clique em “Finish”.



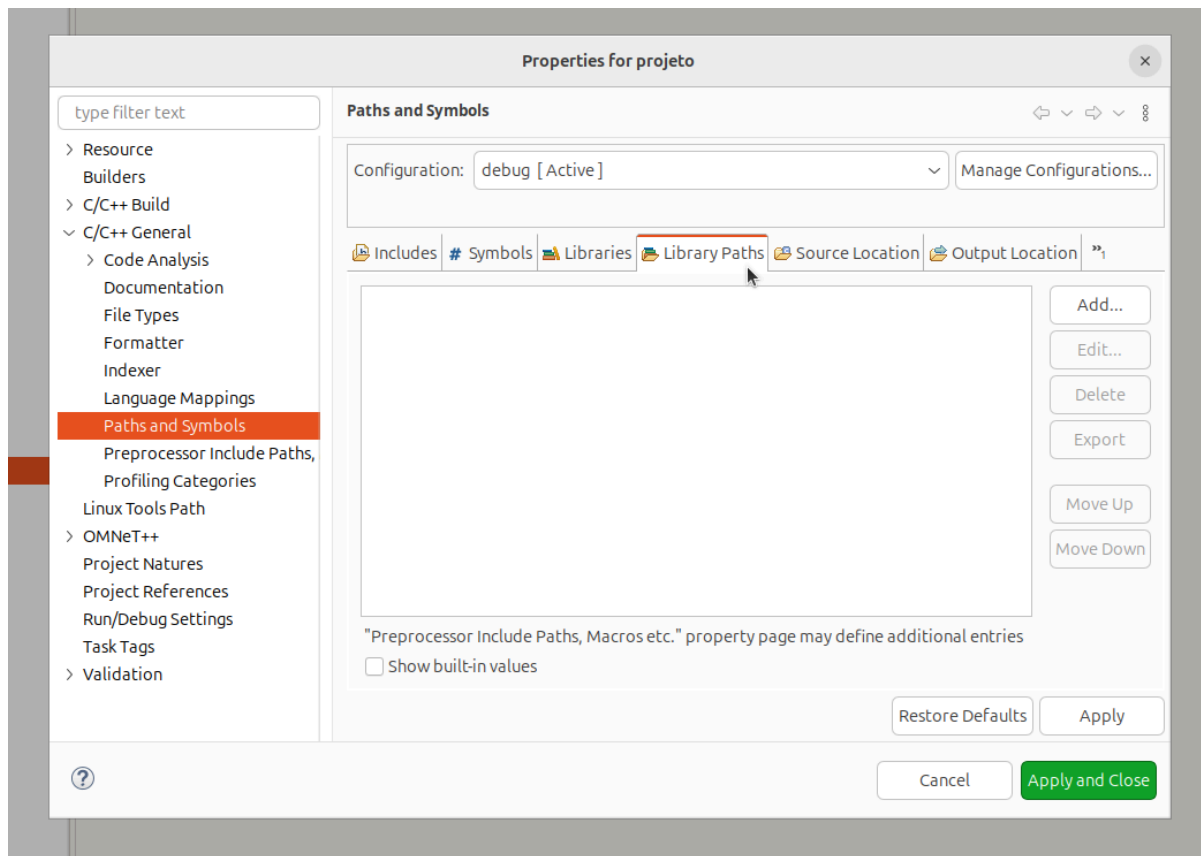
No “Project Explorer”, é possível ver a hierarquia de pastas do projeto:



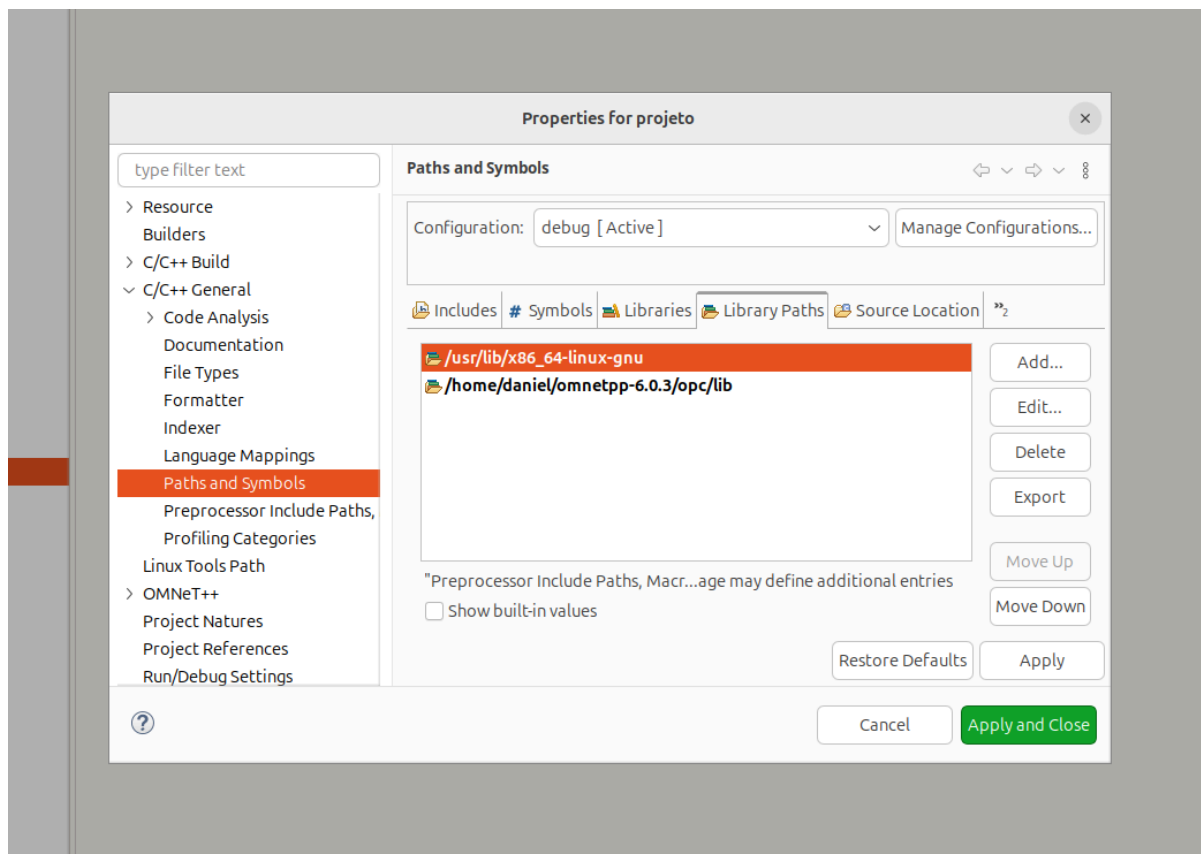
8) Clique com o botão no projeto -> “Properties”



- 9) Acesse 'C/C++ General' -> 'Path and Symbols' -> 'Library Paths'. Em seguida, clique em 'Add'.



10) Adicione os seguintes caminhos:

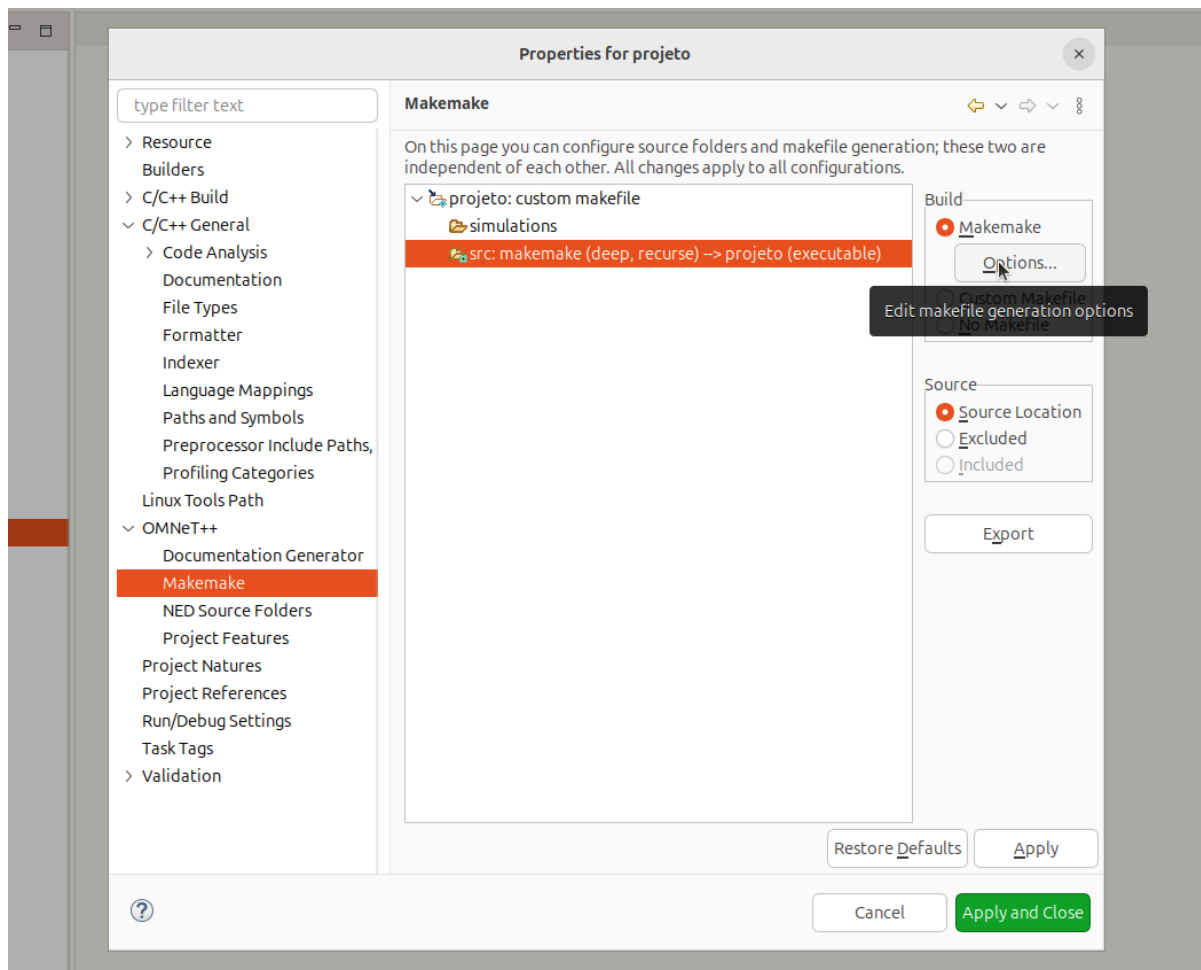


- a) /usr/lib/x86\_64-linux-gnu
- b) /home/user/omnetpp-6.0.3/opc/lib

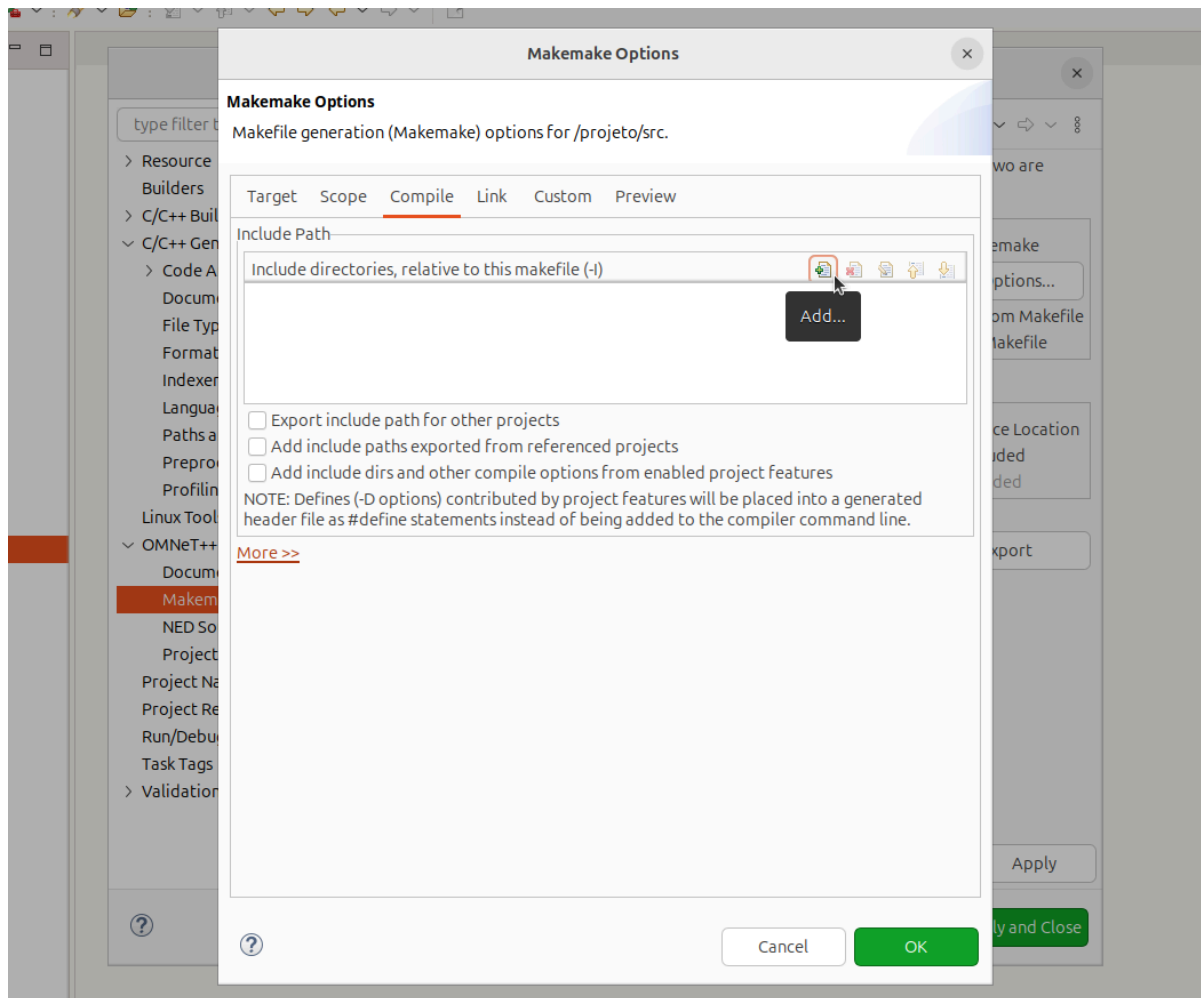
O caminho em **a)** é utilizado para referenciar as bibliotecas terceiras instaladas na etapa **4)**.

O caminho em **b)** aponta para a pasta 'lib' dentro do SDK OPC UA.

- 11) Em seguida, acesse 'OMNeT++' -> 'Makemake' -> selecione 'src: makemake' -> 'Options'.



- 12) Em 'Compile', clique em 'Add'. Adicione os seguintes diretórios:

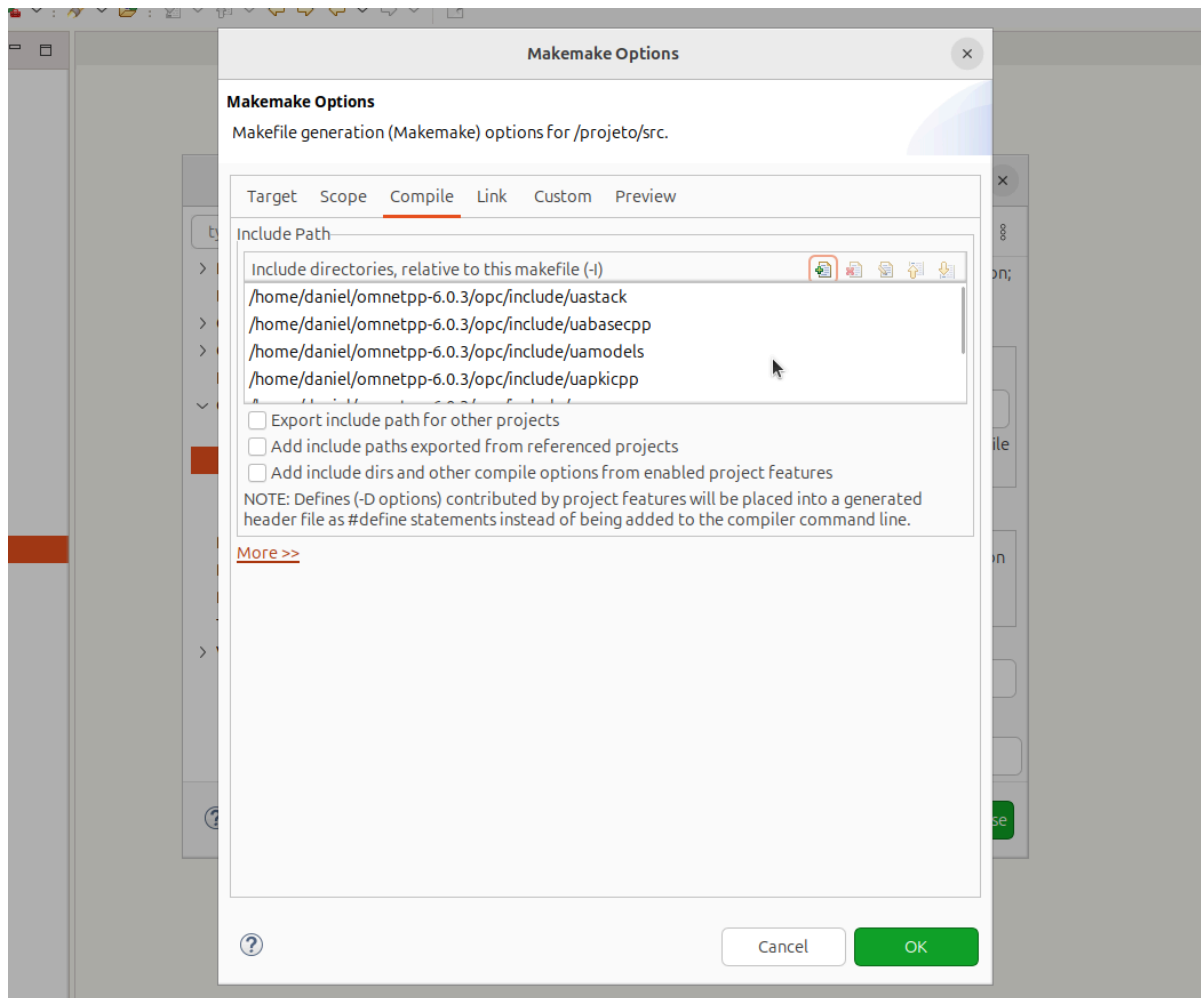


- a) /home/user/omnetpp-6.0.3/opc/include/uastack
- b) /home/user/omnetpp-6.0.3/opc/include/uabasecpp
- c) /home/user/omnetpp-6.0.3/opc/include/uamodels
- d) /home/user/omnetpp-6.0.3/opc/include/uapkcip
- e) /home/user/omnetpp-6.0.3/opc/include/uaservercpp
- f) /home/user/omnetpp-6.0.3/opc/include/xmlparsercpp
- g) /home/user/omnetpp-6.0.3/opc/include/uaclientcpp

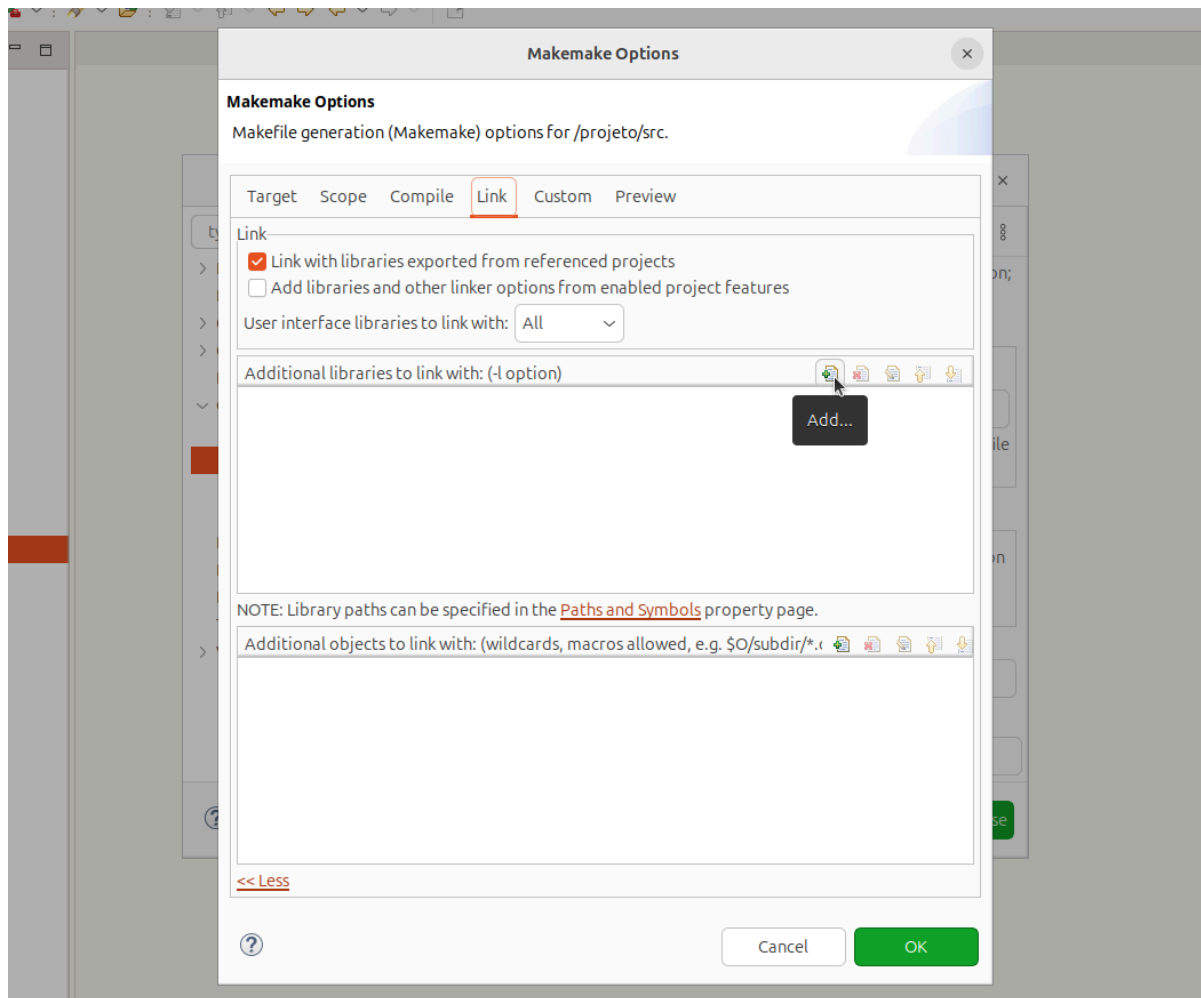
Altere o usuário “user” para o seu.

Os diretórios devem ser adicionados separadamente:



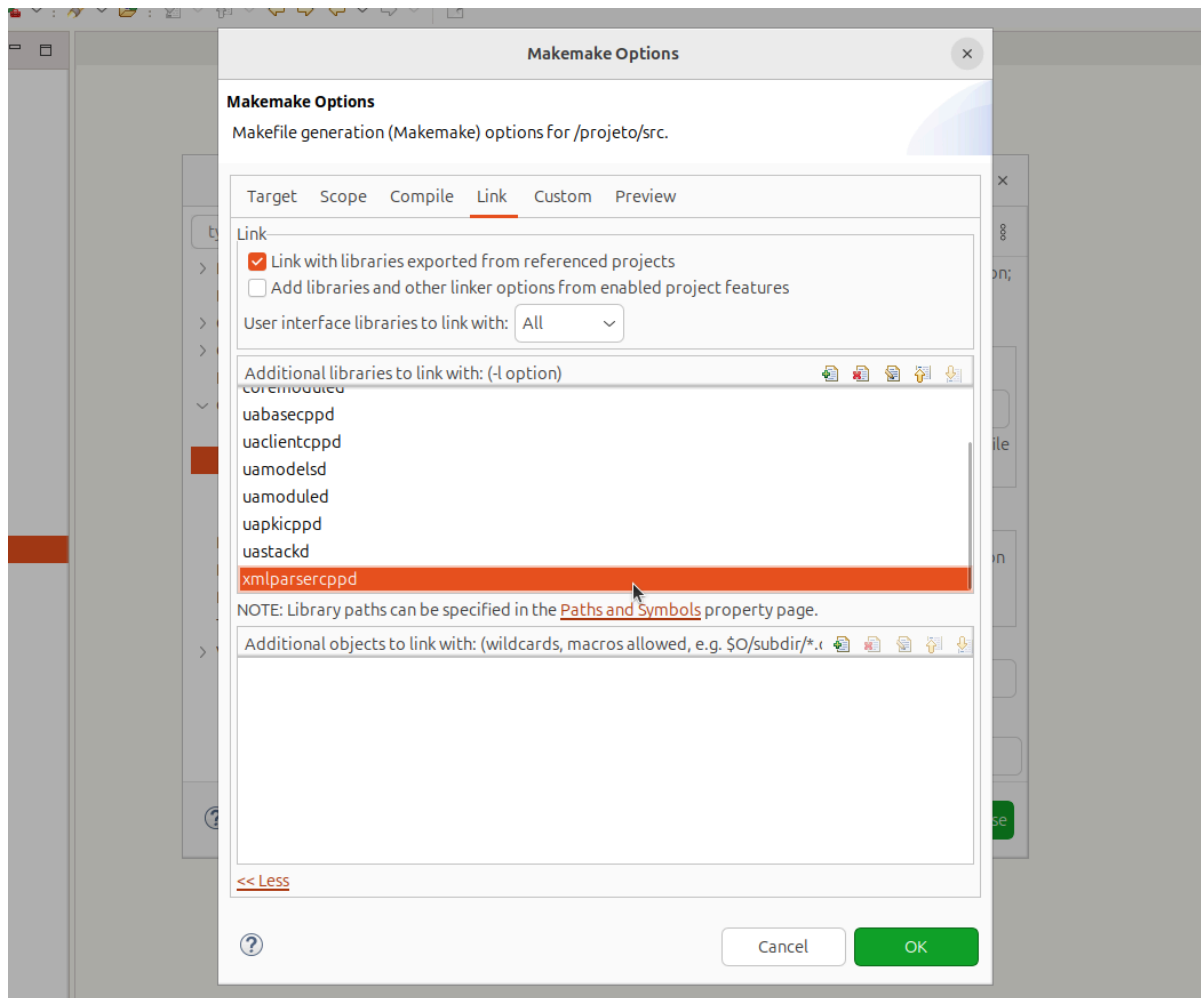


13) Acesse 'Link' -> clique em 'More >>' para expandir as opções -> 'Add'. Adicione as seguintes libraries:



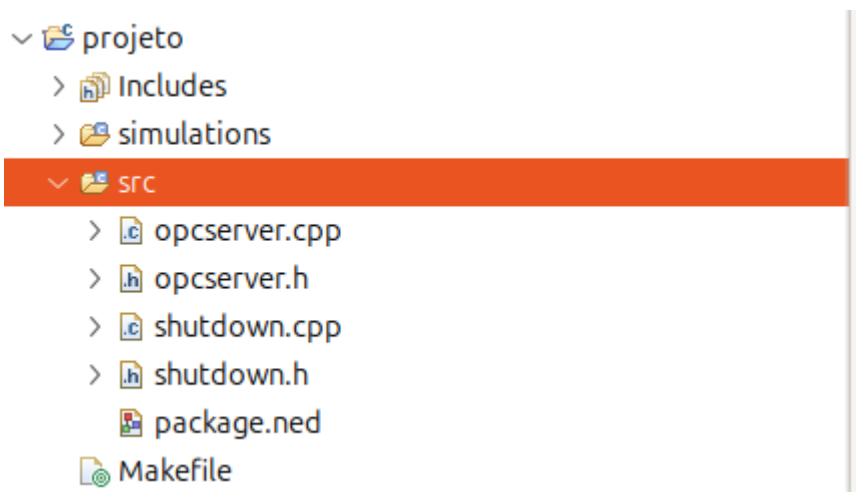
- a) xml2
- b) crypto
- c) coremoduled
- d) uabasecppd
- e) uaclientcppd
- f) uamodelsd
- g) uamoduled
- h) uapkicppd
- i) uastackd
- j) xmlparsercppd

As libraries devem ser adicionadas separadamente:

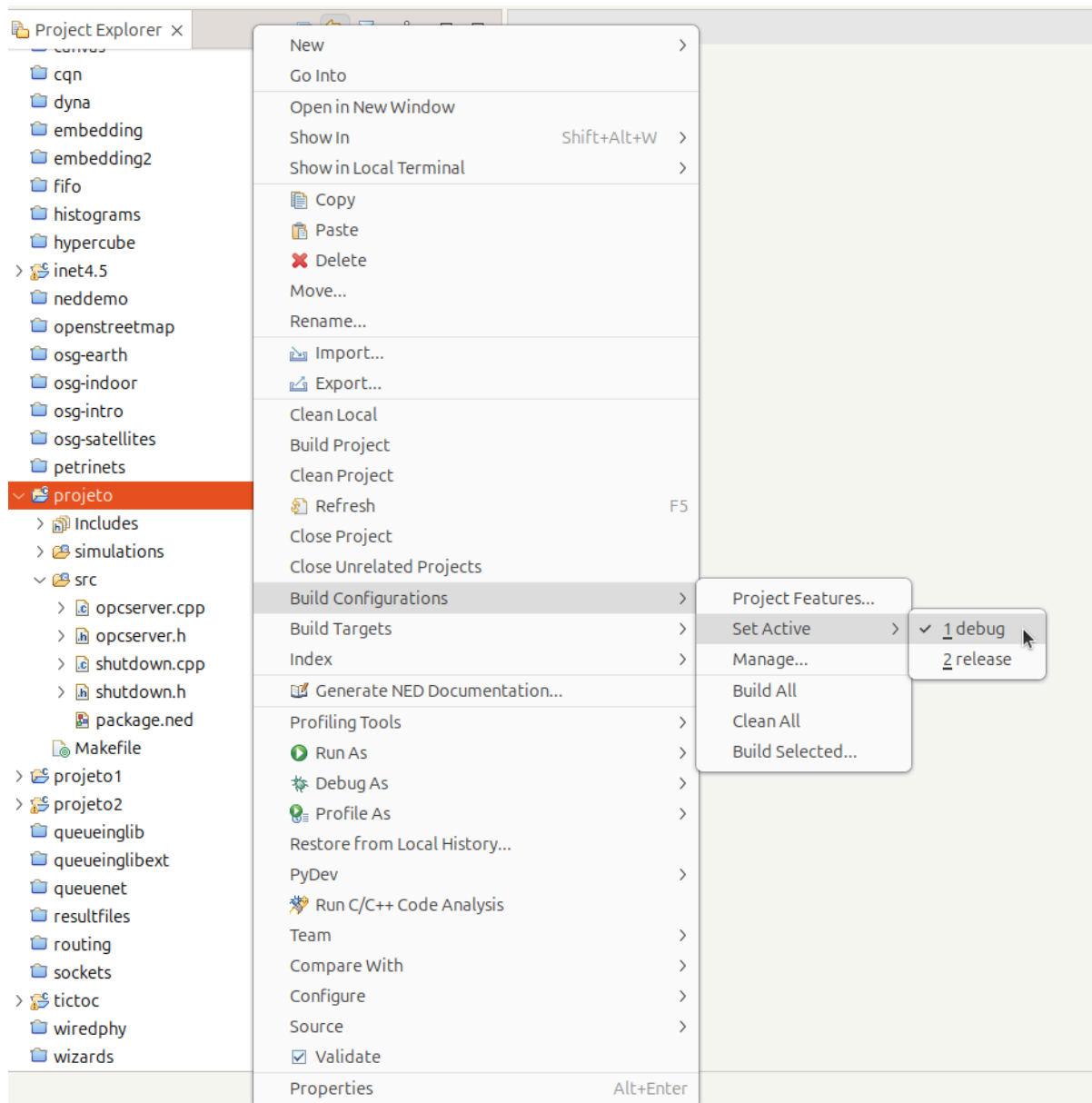


Em seguida, clique em 'OK' -> 'Apply and Close'.

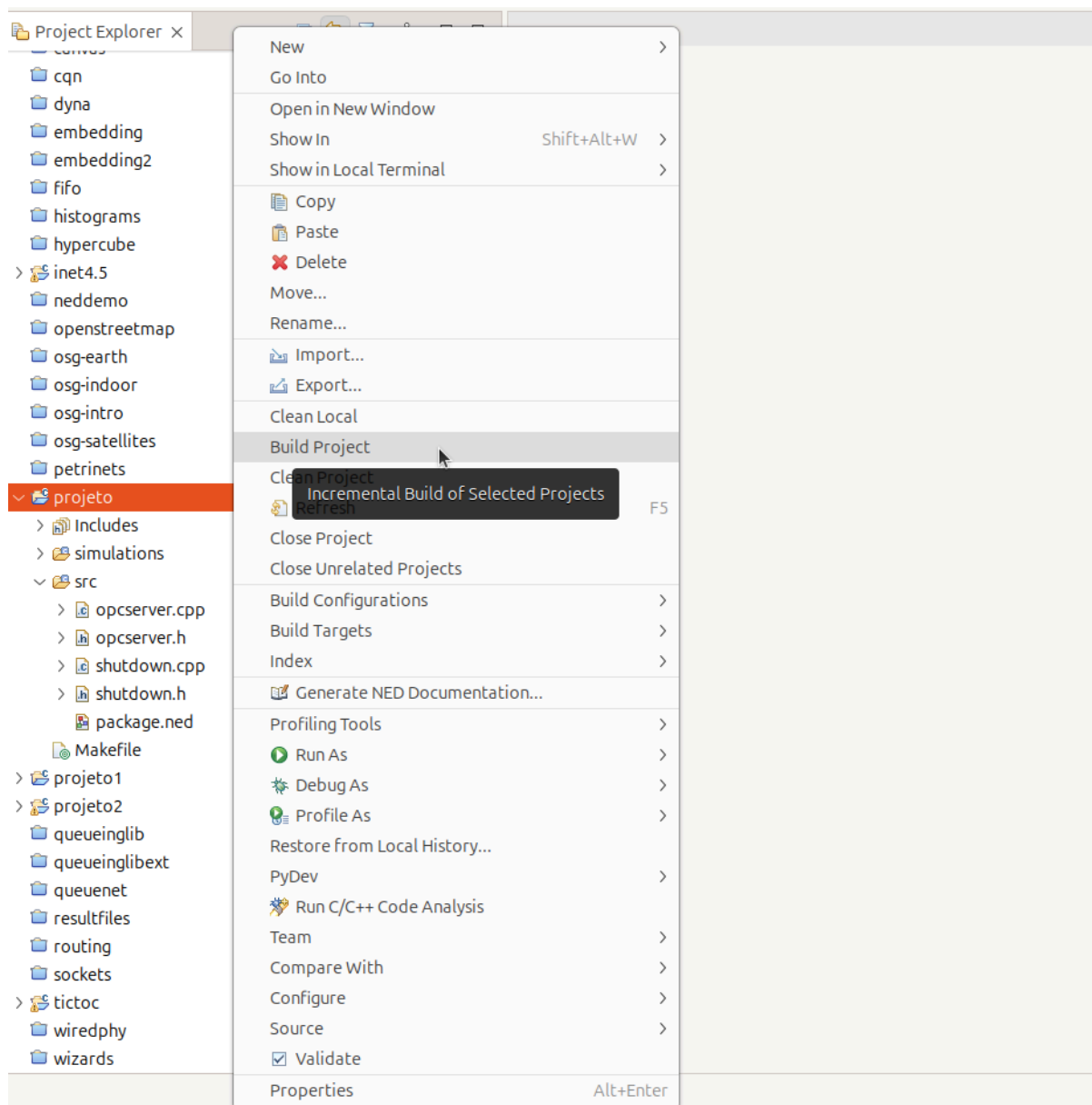
- 14) Copie os arquivos 'opcservice.cpp', 'opcservice.h', 'shutdown.cpp' e 'shutdown.h' localizados em '~/omnetpp-6.0.3/opc/examples/utilities' para a pasta 'src' do projeto.



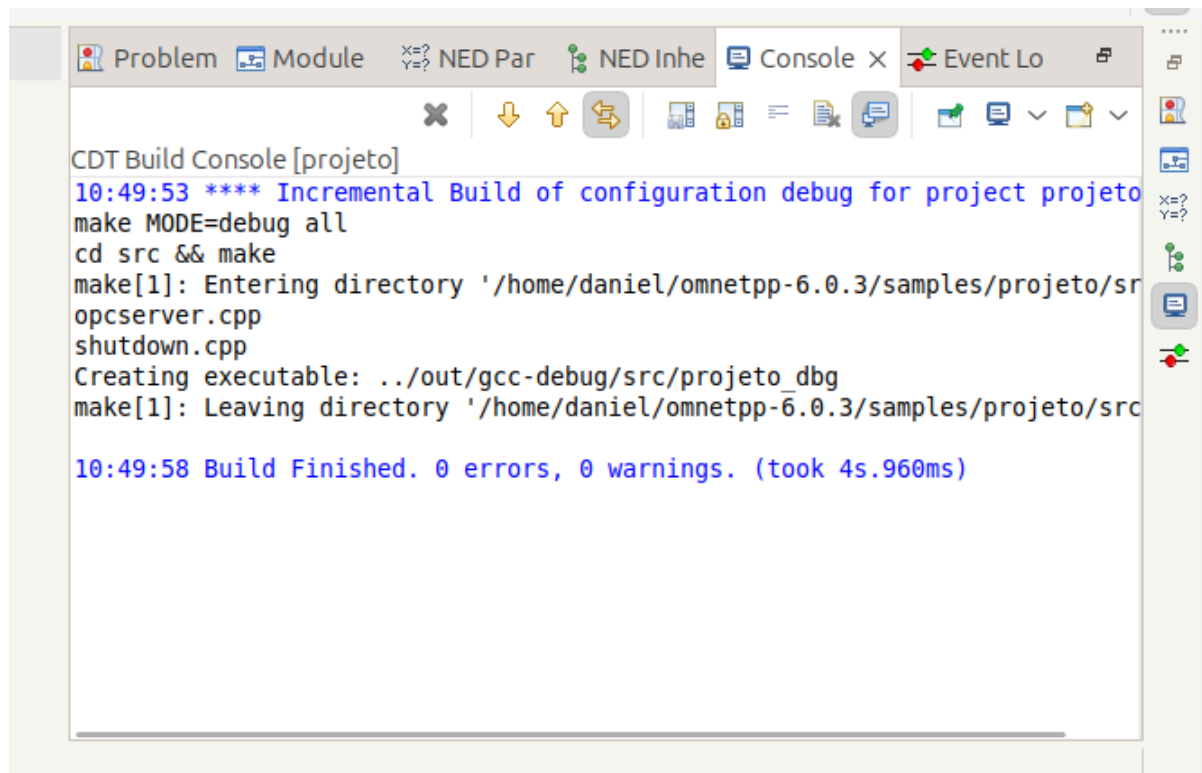
- 15) Clique com o botão direito no projeto, 'Build Configurations' -> 'Set Active' -> 'debug'.



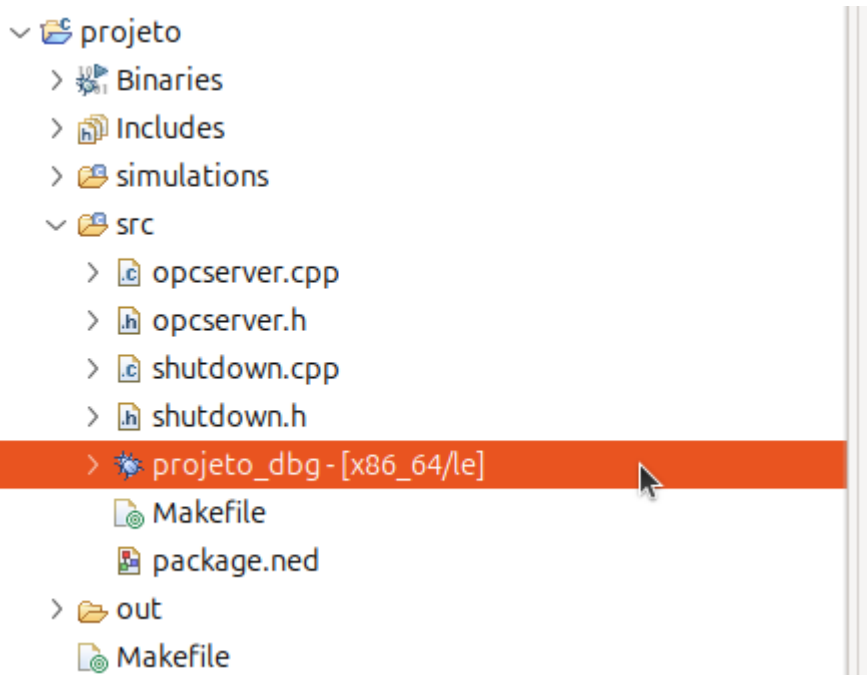
16) Clique em 'Build Project'. Se todos os passos foram seguidos corretamente, a build ocorrerá sem problemas, um arquivo executável e um arquivo Makefile serão criados dentro de 'src'.



17) Saída do terminal:



18) 'projeto\_dbg' e 'Makefile' criados dentro de 'src'.



# Documentação e Construção da Primeira Simulação

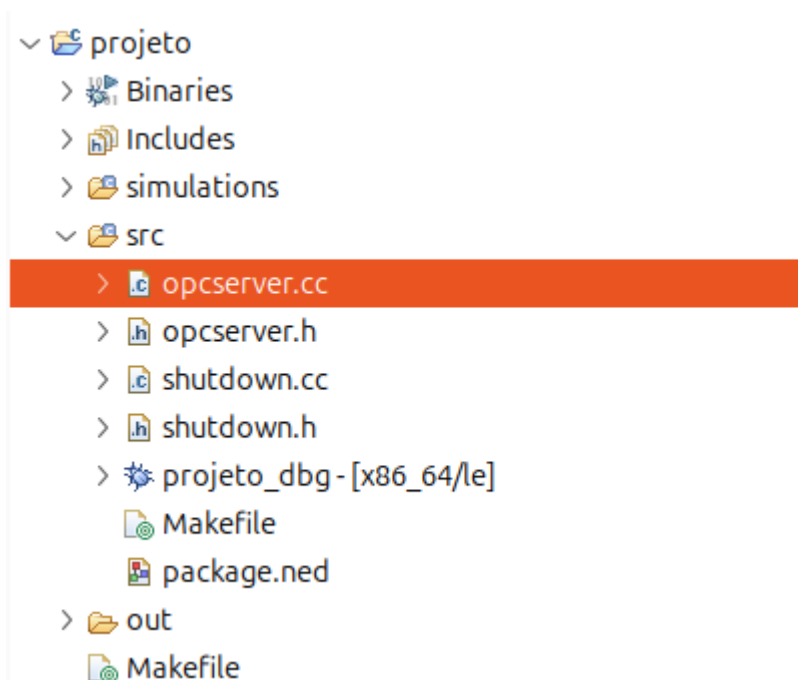
Nesta seção será abordada a criação de uma simulação simples entre um cliente e um servidor via OPC UA. Todo o código fornecido pode ser acessado na branch **master** no repositório no GitHub.

Será utilizada uma abordagem de passo a passo para a construção da simulação, tendo em vista ensinar como funciona a estrutura e hierarquia de pastas do OMNeT++.

Caso haja dúvidas, consulte o TicToc 1 ao 9 na documentação do OMNeT++.

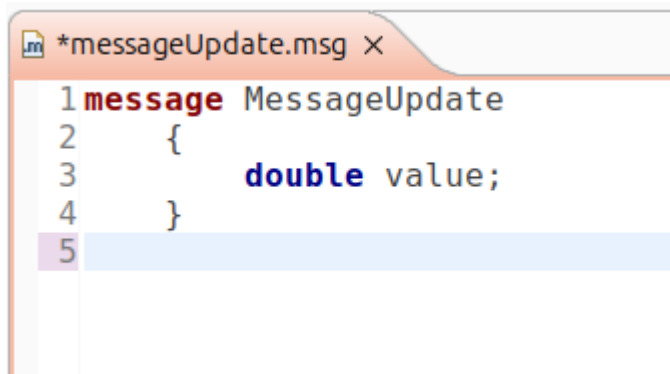
Ideia da simulação: um cliente atualiza uma variável do servidor decrementando seu valor.

- 1) Modifique os nomes dos arquivos 'opcservice.cpp' e 'shutdown.cpp' para 'opcservice.cc' e 'shutdown.cc' respectivamente. Isso é necessário porque o OMNeT++ só aceita um único tipo de extensão de arquivo C++ em um projeto.



- 2) Crie um arquivo chamado 'messageUpdate.msg' na pasta 'src' do projeto com o seguinte conteúdo:

```
message MessageUpdate
{
    double value;
}
```



```
*messageUpdate.msg X
1 message MessageUpdate
2 {
3     double value;
4 }
5
```

Neste arquivo definimos a estrutura da mensagem que enviaremos do cliente para o servidor. Ela pode conter vários atributos de tipos primitivos diferentes. Para o exemplo, usaremos apenas um atributo `'double value'`.

Após a compilação, serão criados dois novos arquivos no projeto: `'messageUpdate_m.h'` e `'messageUpdate_m.cc'`, que implementam funções como `getValue()` e `setValue()`.

- 3) Crie uma classe chamada `'client.cc'` na pasta `'src'` do projeto com o seguinte conteúdo:

```
#include <omnetpp.h>
#include "opcserver.h"
#include "messageUpdate_m.h"

using namespace omnetpp;

class Client : public cSimpleModule
{
    private:
        int messageValue;

    protected:
        virtual void initialize() override;
        virtual void handleMessage(cMessage *msg) override;
};

Define_Module(Client);

void Client::initialize()
{
    messageValue = 10;
    MessageUpdate *msgUp = new MessageUpdate();
    msgUp->setValue(messageValue);
    WATCH(messageValue);
    send(msgUp, "gate$o");
}
```



```

}

void Client::handleMessage(cMessage *msg)
{
    MessageUpdate *msgUp = check_and_cast<MessageUpdate*>(msg);

    EV << msgUp->getValue() << endl;

    msgUp->setValue(--messageValue);
    send(msgUp, "gate$o");
}

```

Nesta classe definimos o cliente. Ao iniciar a simulação, o método `initialize()` é executado primeiro, definindo o valor do atributo `messageValue` para 10 e atribuindo esse valor como o conteúdo da mensagem que será enviada para o servidor.

O método `handleMessage(cMessage *msg)` é responsável por modificar a mensagem recebida do servidor para o cliente. O valor é subtraído de 1 e enviado novamente para o servidor.

- 4) Crie uma classe chamada `server.cc` na pasta `src` do projeto com o seguinte conteúdo:

```

#include "uaplatformlayer.h"
#include "opcserver.h"
#include "shutdown.h"
#ifdef SUPPORT_XML_PARSER
    #include "xmldocument.h"
#endif
#include <omnetpp.h>
#include "opcserver.h"
#include "messageUpdate_m.h"
using namespace omnetpp;
class Server : public cSimpleModule
{
private:
    OpcServer* pServer;
    OpcUa::BaseDataVariableType* pVariable;
protected:
    virtual void initialize() override;
    virtual void handleMessage(cMessage *msg) override;
};

Define_Module(Server);

```

```

void Server::initialize()
{
    char* szAppPath = getAppPath();
    int ret = 0;
    #if SUPPORT_XML_PARSER
        UaXmlDocument::initParser();
    #endif
    ret = UaPlatformLayer::init();
    if (ret == 0) {
        // Create configuration file name
        UaString sConfigFileName(szAppPath);
        #if SUPPORT_XML_PARSER
            sConfigFileName += "/ServerConfig.xml";
        #else
            sConfigFileName += "/ServerConfig.ini";
        #endif
        // Create and initialize server object
        pServer = new OpcServer;
        ret = pServer->setServerConfig(sConfigFileName,
szAppPath);

        if (ret != 0) {
            printf("Error setting OPC server configuration:
%d\n", ret);
            EV << "Error setting OPC server configuration: " +
std::to_string(ret) << std::endl;
        }

        ret = pServer->start();
        if (ret != 0) {
            printf("Error starting OPC server: %d\n", ret);
            EV << "Error starting OPC server: " +
std::to_string(ret) << std::endl;
        } else {
            printf("OPC server started successfully: %d\n",
ret);
            EV << "OPC server started successfully: " +
std::to_string(ret) << std::endl;
        }

        if (ret == 0) {
            NodeManagerConfig* pNodeConfig =
pServer->getDefaultNodeManager();

```

```

        UaVariant defaultValue;
        defaultValue.setDouble(0);

        pVariable = new OpcUa::BaseDataVariableType(
            UaNodeId("Message",
pNodeConfig->getNamespaceIndex()),
            "Message",
            pNodeConfig->getNamespaceIndex(),
            defaultValue,
            OpcUa_AccessLevels_CurrentReadOrWrite,
            pNodeConfig);

pNodeConfig->addNodeAndReference(UaNodeId(OpcUaId_ObjectsFolder, 0), pVariable, OpcUaId_HasComponent);
    }
}

void Server::handleMessage(cMessage *msg)
{
    MessageUpdate *msgUp = check_and_cast<MessageUpdate
*>(msg);

    UaVariant newValue;
    newValue.setDouble(msgUp->getValue());

    UaDataValue dataValue;
    dataValue.setValue(newValue, OpcUa_False, OpcUa_True);

    pVariable->setValue(NULL, dataValue, OpcUa_False);

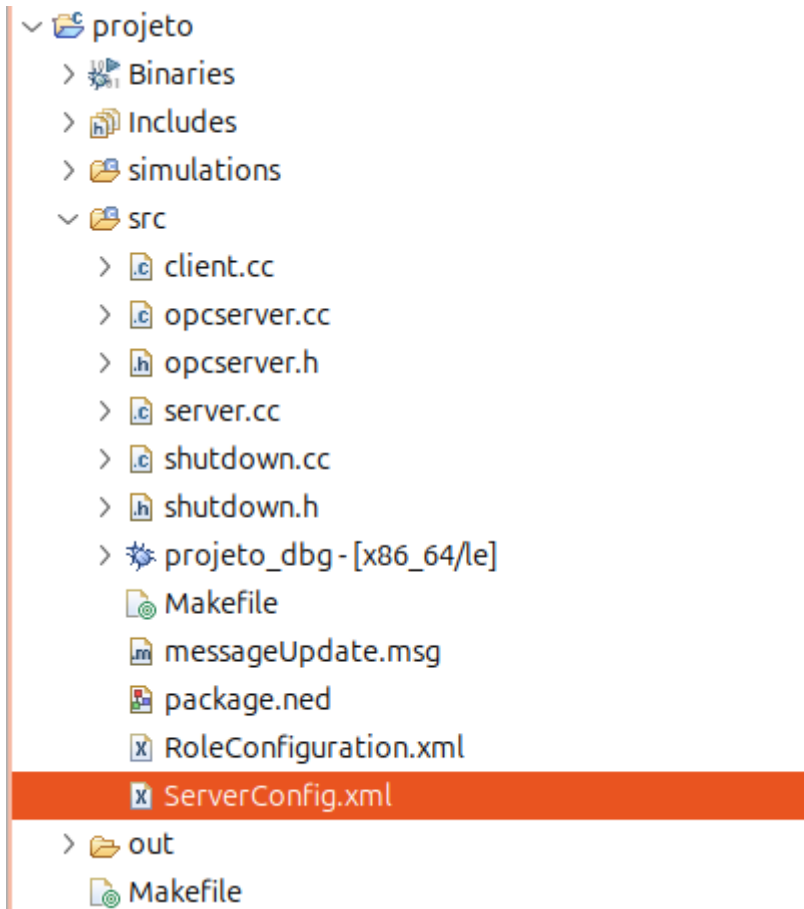
    send(msgUp, "gate$o");
}

```

Esta classe representa o servidor. Nele iniciamos o servidor OPC UA com uma variável chamada 'Message' que possui um valor inicial do tipo double igual a 0.

A função 'handleMessage(cMessage \*msg)' pega o conteúdo da mensagem enviada pelo cliente e sobrescreve o valor da variável no servidor e envia de volta para o cliente.

- 5) Copie os arquivos 'RoleConfiguration.xml' e 'ServerConfig.xml' da pasta '/omnetpp-6.0.3/opc/examples/config' para a pasta 'src' do projeto.



Estes arquivos são necessários para a inicialização e configuração do servidor. Usaremos as configurações padrões disponibilizadas pelo SDK da OPC UA.

- 6) Crie um arquivo chamado 'OpcNetwork.ned' dentro da pasta 'simulations' do projeto com o seguinte código:

```
simple Server
{
    parameters:
        @display("i=block/routing");
    gates:
        inout gate;
}

simple Client
{
    parameters:
        @display("i=block/routing");
    gates:
        inout gate;
}
```

```

network OpcNetwork
{
    submodules:
        server: Server {
            parameters:
                @display("i=,purple");
        }

        clientA: Client {
            parameters:
                @display("i=,cyan");
        }

    connections:
        clientA.gate <--> { delay = 100ms; } <--> server.gate;
}

```

Arquivos .ned são utilizados pelo OMNeT++ para configurar as conexões e topologias entre clientes e servidores. No exemplo, o cliente se conecta ao servidor com um delay de 100ms.

7) Acesse o arquivo 'omnetpp.ini' na pasta 'simulations' e insira o seguinte conteúdo:

#### [General]

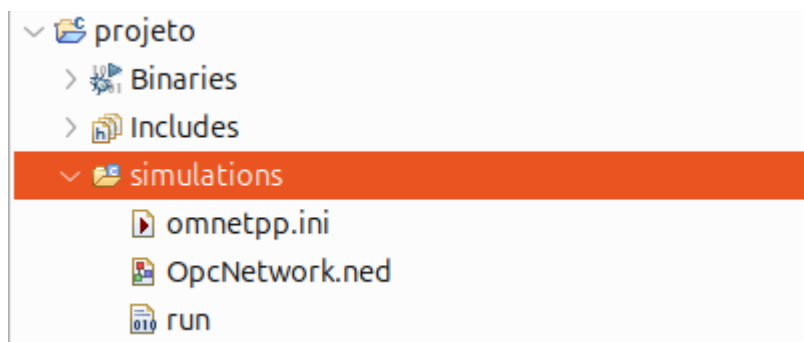
```

network = OpcNetwork
record-eventlog = true
sim-time-limit = 5s

```

Arquivos .ini são usados para configurações de inicialização das simulações, como tempo limite e se o eventlog será gravado.

Ao final, a pasta 'simulations' deve ter o seguinte conteúdo:



8) Realize a build do projeto. Para a execução, cada branch possui um script 'run\_simulation' na pasta 'simulations', basta executá-lo no terminal:

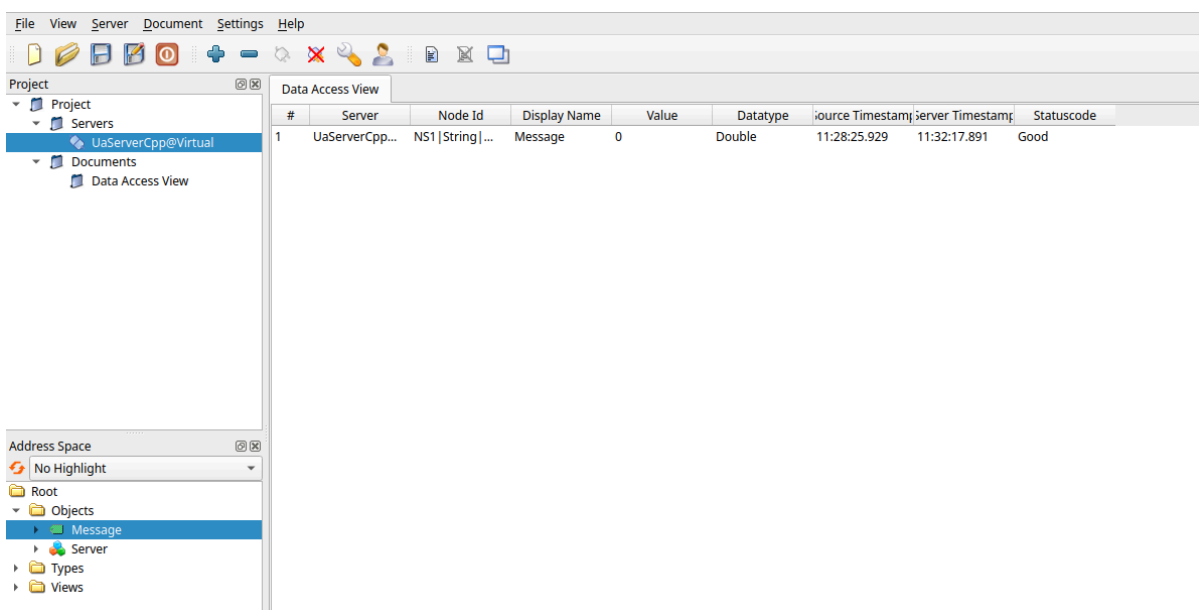
```
daniel@Virtual: ~/omnetpp-6.0.3/samples/projeto/simulations
daniel@Virtual:~/omnetpp-6.0.3/samples/projeto/simulations$ ./run_simulation
```

Caso não haja erros, o output do terminal será o seguinte:

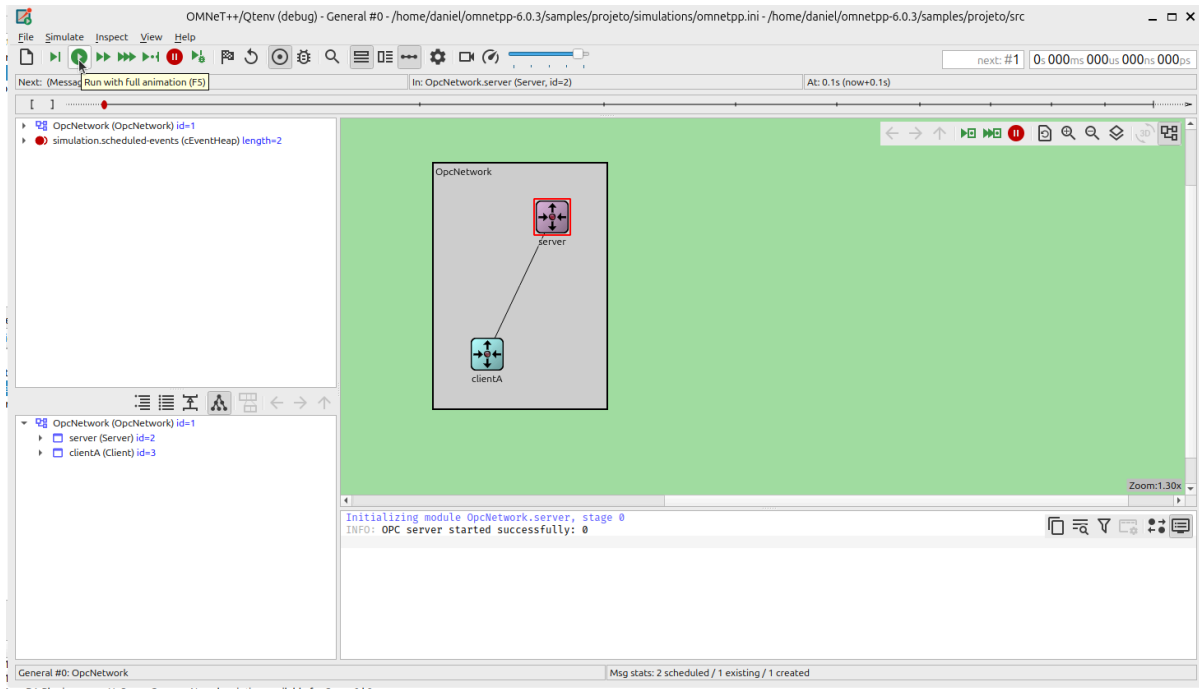
```
QSocketNotifier: Can only be used with threads started with QThread
MESA: error: ZINK: failed to choose pdev
libEGL warning: egl: failed to create dri2 screen
qt.qpa.wayland: Wayland does not support QWindow::requestActivate()
Recording eventlog to file '/home/daniel/omnetpp-6.0.3/samples/projeto/simulations/results/General-#0.eelog'...
*****
Server opened endpoints for following URLs:
    opc.tcp://Virtual:48010
*****
OPC server started successfully: 0
qt.qpa.wayland: Wayland does not support QWindow::requestActivate()

*****
The OPC UA server SDK is running in demo mode.
Communication will be stopped in 60 minutes.
*****
```

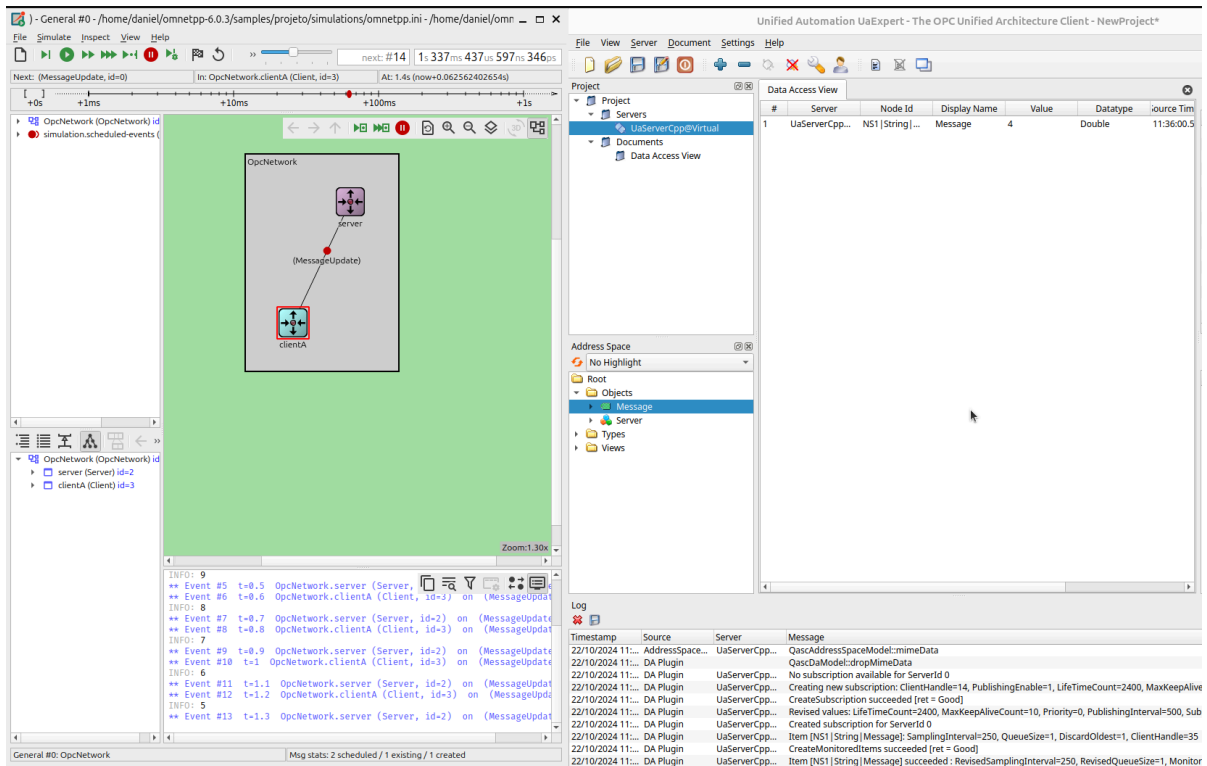
9) Copie o endereço 'opc.tcp://Virtual:48010' e inicie uma conexão no UaExpert:



## 10) Inicie a simulação:



É possível observar o valor da variável mudar no UaExpert a cada interação do cliente com o servidor:



# Documentação da Simulação entre 5 clientes e 1 servidor

A documentação a seguir se baseará no código da branch **randomUpdates** no repositório do GitHub.

Caso haja dúvidas, consulte o TicToc 10 em diante na documentação do OMNeT++.

Ideia da simulação: para cada cliente, existe uma variável do tipo double no servidor que será atualizada a partir de cada interação cliente-servidor randomicamente.

## client.cc

- No método `initialize()`, o atributo `messageValue` é inicializado com um valor double aleatório gerado pela função `generateValue()`, em seguida criamos uma nova mensagem do tipo `MessageUpdate` onde atribuímos o valor da mensagem como o atributo `messageValue`. Logo após, definimos o index do cliente que enviamos a mensagem a partir da função `send()`.
- No método `handleMessage()`, primeiro inicializamos uma nova mensagem do tipo `MessageUpdate` a partir da mensagem recebida por meio de `check_and_cast` (procedimento recomendado pelo OMNeT++), em seguida atualizamos o valor do atributo `messageValue` com outro valor aleatório e atribuímos na mensagem. Após isso, transmitimos novamente a partir da função `send()`. Dessa forma, ao receber uma resposta o cliente atualiza o valor da mensagem recebida e envia novamente para o servidor.
- O método `generateValue()` retorna um número aleatório do tipo double entre 0.0 e 100.0.

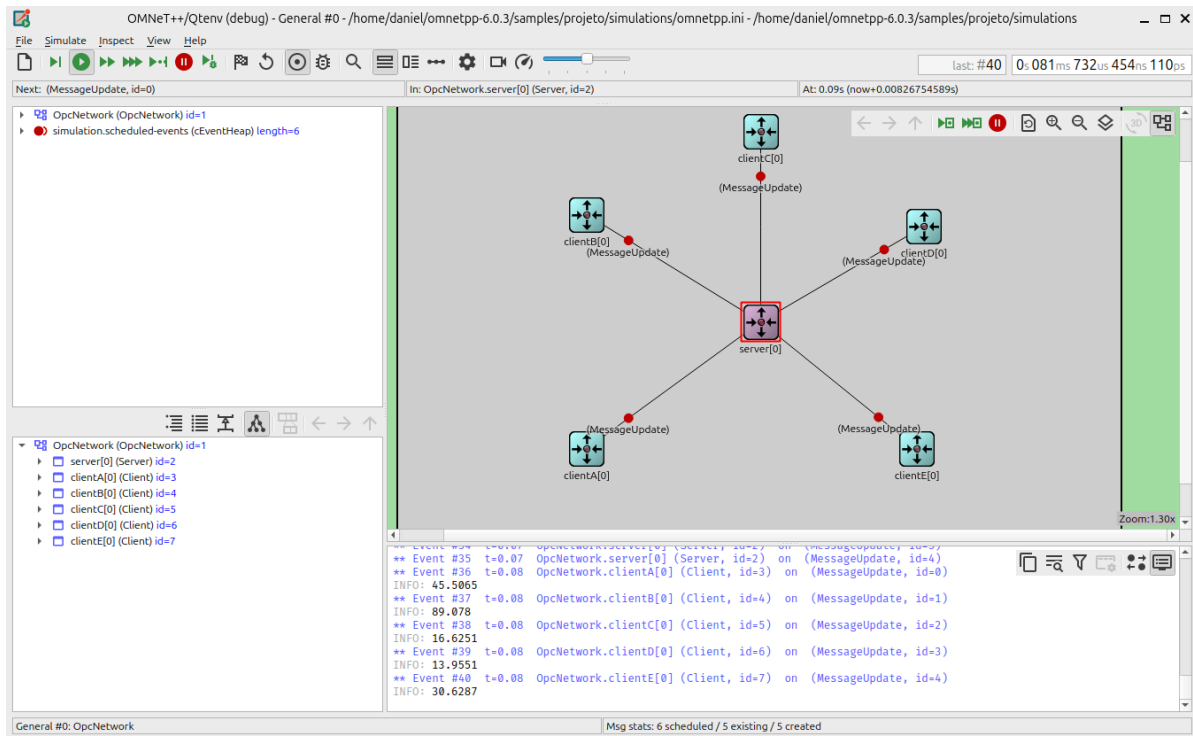
## server.cc

- Adicionamos um array `listaVariaveis[5]` do tipo `BaseDataVariableType` na linha 19. Este será utilizado para armazenar os 5 nós para cada cliente.
- Na linha 68 adicionamos um `for (int i = 0; i < 5; i++)` para percorrer o array e atribuir o valor na linha 86.
- No método `handleMessage(cMessage *msg)` na linha 96 buscamos o index do cliente que enviou a mensagem e a partir desse número acessamos o array na linha 104 e definimos o novo valor.

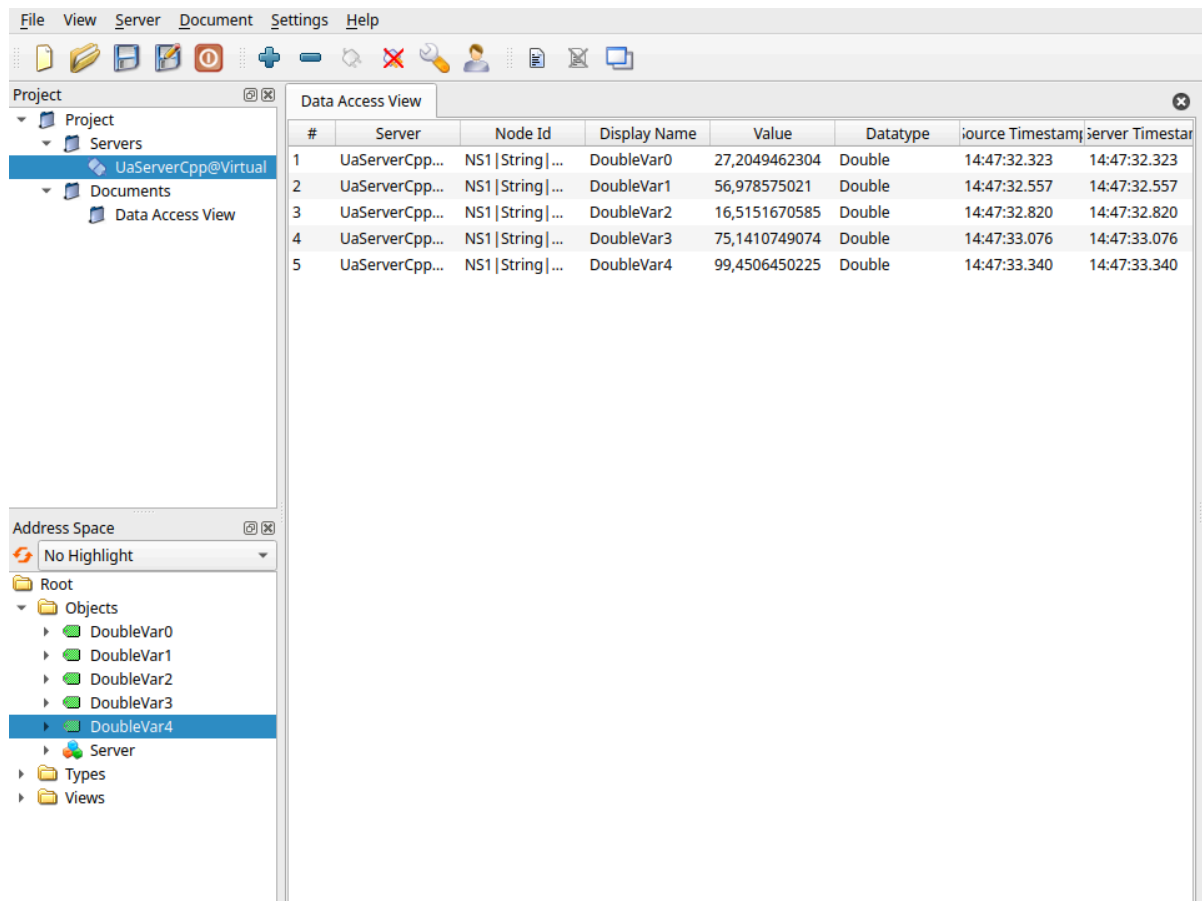
## OpcNetwork.ned



- Nas linhas 6 e 14 mudamos alteramos as portas para vetores para que seja possível a conexão de vários clientes.
- Na linha 21 em diante adicionamos toda a lógica da conexão entre os 5 clientes com 1 servidor.
- Após iniciar a simulação, cada cliente vai gerar um novo valor e enviar para o servidor repetidamente:



Ao realizar a conexão com o UaExpert, é possível observar o valor das variáveis mudar:



## Documentação da Simulação entre dois hosts e um switch utilizando o framework INET sem conexão OPC

A documentação a seguir se baseará no código da branch **INET** no repositório do GitHub.

Ideia da simulação: todos os nós da rede serão baseados no framework INET, com dois hosts e um switch intermediário. Por enquanto, não serão modificados arquivos C++, apenas arquivos .ned e .ini.

### OpcNetwork.ned

- Importações necessárias realizadas nas linhas 1 a 4.
- Definição dos hosts e demais componentes necessários nas linhas 34 a 44.
- Conexão entre 'host <--> switch <--> host' nas linhas 48 e 49.

### omnetpp.ini

- Nas linhas 6 a 16 do arquivo de inicialização, foram definidos alguns parâmetros obrigatórios para simulações utilizando o framework INET.
- Compile e rode a simulação utilizando o script 'run\_simulation'.

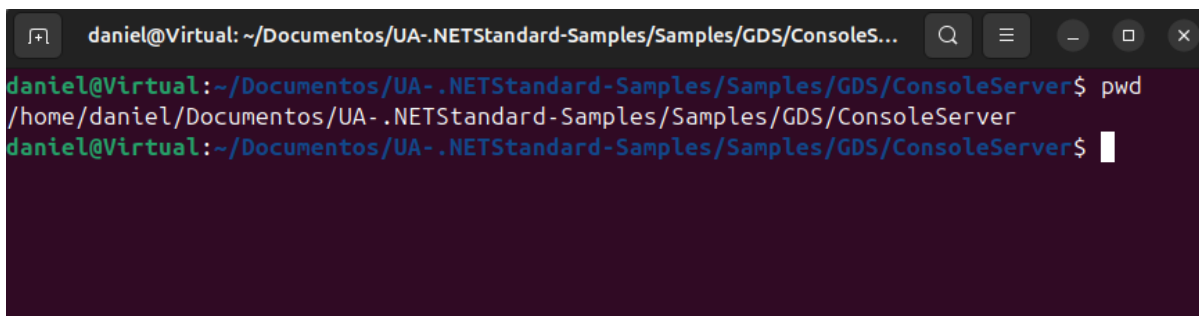
## Considerações

- Algumas linhas de código no arquivo OpcNetwork.ned foram comentadas de tentativas anteriores de iniciar uma simulação utilizando OPC, porém sem sucesso. Isso será abordado mais adiante.
- O código da simulação foi retirado de um exemplo do framework. Pode ser acessado em "inet4.5/examples/ethernet/vlan".

## Integração com GDS (Global Discovery Server)

Ideia da simulação: conectar o GDS com o UaExpert.

- 1) Instale a versão 8 do .NET no ambiente Linux.
- 2) Acesse o [repositório](#) do GDS disponibilizado pela OPC Foundation e realize o seu download.
- 3) Na raiz do projeto, acesse "/Samples/GDS/ConsoleServer/":



```
daniel@Virtual: ~/Documentos/UA-.NETStandard-Samples/Samples/GDS/ConsoleS...
daniel@Virtual:~/Documentos/UA-.NETStandard-Samples/Samples/GDS/ConsoleServer$ pwd
/home/daniel/Documentos/UA-.NETStandard-Samples/Samples/GDS/ConsoleServer
daniel@Virtual:~/Documentos/UA-.NETStandard-Samples/Samples/GDS/ConsoleServer$
```

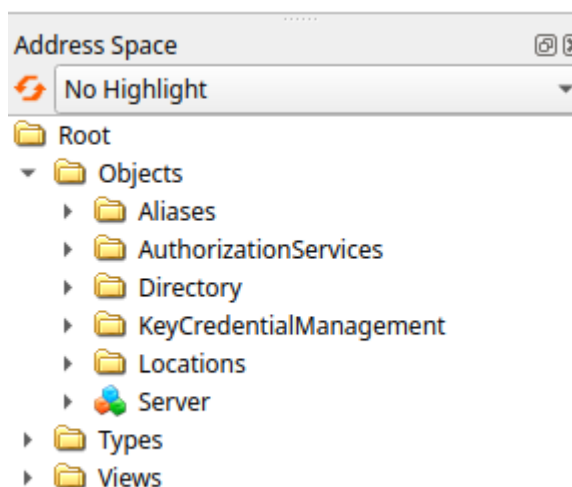
- 4) Execute o comando "dotnet run". Caso seja a primeira vez, o .NET irá iniciar o processo de build das dependências.
- 5) Após a build, escolha "default users" pressionando "y".
- 6) Copie o endereço fornecido:

```
daniel@Virtual: ~/Documentos/UA-.NETStandard-Samples/Samples/GDS/ConsoleS...
daniel@Virtual:~/Documentos/UA-.NETStandard-Samples/Samples/GDS/ConsoleServer$ pwd
/home/daniel/Documentos/UA-.NETStandard-Samples/Samples/GDS/ConsoleServer
daniel@Virtual:~/Documentos/UA-.NETStandard-Samples/Samples/GDS/ConsoleServer$ dotnet
run
.Net Core OPC UA Global Discovery Server
Use default users? (y/n, default y): y
opc.tcp://virtual:58810/GlobalDiscoveryServer
Server started. Press Ctrl-C to exit...
```

- 7) Acesse o UaExpert, realize a conexão em “Custom Discovery” com o endereço copiado e escolha qualquer uma das abordagens de segurança:



- 8) Observe o Address Space do GDS:



# Dificuldades e Próximos Passos

Aqui, abordarei as principais dificuldades encontradas e os próximos passos a serem seguidos na pesquisa.

## Métodos OPC

Com as simulações atuais, como as presentes na branch **master** e **randomUpdates**, a alteração das variáveis no servidor é feita de forma direta em “baixo nível”, utilizando um array para armazenar as variáveis e um ‘for’ para percorrer todas e atualizar individualmente. Este não é o modo padrão OPC que queremos. Para isso, é necessário utilizar métodos OPC UA disponíveis no SDK da Unified Automation para realizar tanto a criação, como modificação e manutenção dessas variáveis. Desse modo, se faz necessário o estudo do framework OPC utilizado a fim de descobrir como realizar essa comunicação de forma efetiva. A utilização do array para as variáveis foi uma abordagem de teste que funcionou mas acredito que não seja a maneira correta tendo em vista os recursos do SDK.

## INET + OPC UA

Até o presente momento, não encontrei uma abordagem que seja possível utilizar os nós e switches do framework INET juntamente com o código criado para a inicialização e comunicação do servidor OPC. Encontrei alguns erros de herança e sobrescrita de métodos durante a integração, devido ao fato do INET utilizar um código próprio em C++ e em .ned para os seus componentes. No caso, seria necessário alterar os métodos do framework por meio de overrides ou outra forma que possa estar presente na documentação.

A seguinte [documentação](#) do INET (não muito clara, infelizmente) descreve a utilização de FieldsChunks em classes do tipo .msg em OMNeT++ para realizar uma comunicação customizada entre os componentes do framework. Acredito que essa seja a melhor maneira de enviar uma mensagem OPC entre um hosts e switches INET.

## Simulação de mais de um servidor

O framework OPC UA da Unified Automation não permite iniciar no mesmo processo mais de um servidor. Como as simulações executam na mesma instância de terminal que o servidor é iniciado, não é possível ter mais de um servidor ao mesmo tempo. Acredito que esse seja um dos maiores problemas a serem resolvidos. Pensei na abordagem de programação concorrente simulando cada instância de um servidor como um processo à parte por meio de ‘fork()’ porém não obtive êxito, mas acredito que seja possível. Talvez no próprio SDK ou fóruns da Unified Automation esse assunto seja abordado.

## Conexão com GDS

Não obtive muitos avanços com o GDS abordado anteriormente, apenas a conexão com o UaExpert mas nenhuma ligação com as simulações realizadas no OMNeT++. É necessário pesquisar no repositório oficial como realizar a conexão definitiva.

Outra nuance percebida foi a versão do .NET. Existem outras aplicações no repositório, como “Client”, “ClientControls” e “ConsoleServer”. Não é possível realizar build de nenhuma das três porque é necessária a versão 4.8 do .NET, que só está disponível para ambientes Windows. Uma solução seria testar a build no sistema operacional da Microsoft e migrar toda a aplicação OMNeT++ junta.